UNIVERSITAT DE BARCELONA

FUNDAMENTAL PRINCIPLES OF DATA SCIENCE MASTER'S THESIS

Facing the Label-Switching problem when using generic inference platforms for crowd annotation models

Author:
Àlex PADRÓS ZAMORA

Supervisors:
Jerónimo HERNÁNDEZ-GONZÁLEZ
Jesús CERQUIDES

A thesis submitted in partial fulfillment of the requirements for the degree of MSc in Fundamental Principles of Data Science

in the

Facultat de Matemàtiques i Informàtica

June 30, 2021

UNIVERSITAT DE BARCELONA

Abstract

Facultat de Matemàtiques i Informàtica

MSc

Facing the Label-Switching problem when using generic inference platforms for crowd annotation models

by Àlex PADRÓS ZAMORA

In this Master Thesis we study some classical approaches for crowd annotation models such as the pooled multinomial model or the Dawid-Skene models. These models try to learn from the crowd, which is not required to be composed of experts. In particular, the problem of label aggregation that we deal with can be seen as a probabilistic graphical model. We propose an algorithm that aims to solve the problem of label-switching for generic inference platforms such as STAN without any previous intervention to the optimization/sampling method. We also study its performance by means of the Kullback-Leibler divergence, where we see that the results are better after applying our proposed correction.

Acknowledgements

This has been a complicated and strange year. I arrived to the first class of the course without having written a single line in Python (almost 90% of the course has been taught in Python). Moreover, I started working in a big company (for the first time in my life) almost at the same time I started the course. Due to these two reasons, first of all, I would like to thank my family and my classmates. To my mom and sister for all the support they have given me during this year full of novelties for me, where the battle against stress has been a constant and mental health has been difficult to take care of, due to the mobility restrictions caused by Covid, among others. They have made it look easy, when it is really not. I would like to thank my classmates for repeatedly taking time out of their lives (which I am sure they were also very busy) to help me clarify doubts. Without this little help, I am aware that everything would have been much more complicated. I would like to give a special mention to my university buddy and friend Aurelio, who has been an inspiration to me over the years, showing me that with passion, dedication and sacrifice you can achieve whatever you set your mind to, even when it seems impossible.

I would also like to thank my advisor Jerónimo for his recommendation to do this thesis. At first, I had chosen to do a different one about Recommenders Systems, since I am also very interested in them. However, he thought that I would enjoy this project because of all the mathematics behind it, since it is my background. I would also like to thank my other advisor Jesús for all the help and support with this thesis week after week. I am sure they will both do great in their careers.

I would also like to take this opportunity to thank my girlfriend Cristina for all the support she has given me over the years. As I mentioned before, mental health is essential and it is difficult to take care of without adequate support. I find that in this aspect, I have been very fortunate.

To finish this acknowledgments, I would like to thank my lifelong friends "Marshall Wheel", Marc and Marina for their great friendship and support in difficult times. I am sure that they are as excited as I am to see them appear in this text and that they will love to read this work.

Contents

Chapter 1

Introduction

Information technologies have grown almost immeasurably in recent years. As an advantage, they have allowed to begin another type of science: the citizen science, which takes advantage of the power of masses and is used to efficiently complete scientific tasks. Crowdsourcing approaches to machine learning aim to obtain data from a community of collaborators who are not required to be experts or knowledgeable in the project in which they participate. Commonly, the type of task assigned to the crowd is not difficult, and is usually based on the labeling of images, audios... Different methods have been proposed for learning from this type of data during the last years. These methods focus mainly on solving the problem of label aggregation (combining different opinions for the same data point) for machine learning. With enough workers, several studies have shown that one can match (in terms of reliability) the accuracy that would have been given by a smaller set of experts of the task at hand.

In this work, we will study a few classical approaches for crowdsourcing problems. The first one is the pooled multinomial model (3.2.3). This model is quite simple, since it assumes that all the annotators share the same ability for labeling. In order to see a more general version, we will study the Dawid-Skene models (3.2.3). These models distinguish between annotators, and try to model each of them. We will see three versions: the general one, the conditional and the homogeneous. As its name indicates, the first one is the most general: if the number of different labels is k, then the number of free parameters to model is $k(k-1)$, which can easily lead us to overfit on small datasets. The other two are simplifications of the general one that give some restrictions to the modeled behavior of the annotators. The number of free parameters to model for these models are k and 1, respectively.

Chapter 2 is devoted to see some background concepts that are necessary to follow this thesis. We will give an introduction to Bayesian Statistics, comparing it to the classical frequentist approach. Our probabilistic programs will follow a Bayesian pipeline. We will also introduce what a probabilistic graphical model is, even though we will only talk about Bayesian Networks, since they are the ones we are interested in for our particular problem. To conclude the chapter, we will define the Kullback-Leibler divergence, which is basically a measure of how different two probability distributions are from each other.

Chapter 3 will present our problem. Firstly, we will give an historical introduction to crowdsourcing, defining in detail what each concept is. After that, we will define our problem, both conceptual and mathematically. We will actually see that the problem of label aggregation can be modeled as a Bayesian Network, which will help us to clarify how to factorize our probability distributions. This will help us to better understand the pooled multinomial and the Dawid-Skene models. To finish the chapter, we will present the inference platform that we are using: STAN. We will explain *L-BFGS* (3.3.1), the optimization method that STAN uses by default and the

one we will use for our experiments. We will also see an important drawback of STAN: the lack of support for discrete parameters. In Chapter 4, once the theory is well studied, we will start focusing on the main point of this thesis, the label-switching problem (4.1). In this chapter, we will see a theoretical point of view and a proposal to correct it.

We will present the results of our experiments in Chapter 5. Before starting, we will explain the (theoretical) discrete parameter marginalization process that we have had to perform in order to write our pooled multinomial model in STAN. Other processes that we use have been previously described in (3.2.3). After that, we will present an algorithm that aims to solve the label-switching problem, under the hypothesis that our annotators are mostly good (i.e. they label tasks correctly, in general). Our goal is to correct the label-switching problem without previously modifying (or minimally modifying) the input of the STAN optimization algorithms. In order to test our proposal, we will work with many kinds of datasets. Some of them will be created intentionally to make it difficult for our algorithm. For instance, we will work with a set of annotators that do not fulfill our hypothesis, testing thus our method in difficult experimental scenarios. Another example will be a dataset where the vast majority of tasks will be of a particular class. In general, we will see that the results (in terms of Kullback-Leibler divergence) after applying the label-switching correction algorithm are much better.

Chapter 2

Background

In this chapter, we will present some background concepts that need to be understood in order to follow this work.

2.1 Bayesian approach and classical theorems

The aim of this section is to understand what is the difference between classical probability (i.e. frequentist approach) and the Bayesian approach (Eddy, 2004; Downey, 2015), as well as to explain the necessary background concepts.

The principal concept that we must define is the notion of conditional probability:

Definition 2.1.1. ***(Conditional probability)*** *Given two events A and B with $\mathbb{P}(B) > 0$, the conditional probability of A given B, which is denoted by $\mathbb{P}(A|B)$, is given by*

$$\mathbb{P}(A|B) := \frac{\mathbb{P}(A,B)}{\mathbb{P}(B)}.$$

Given a discrete random variable[1] X, we define its *probability mass function (pmf)* as the function that maps each value x_j of X to its associated probability (i.e. $\mathbb{P}(X = x_j) = d_j$) and 0 to the rest of real numbers. More formally, if X takes values $x_1 < \cdots < x_m$:

$$\begin{aligned} \text{pmf}_X : \mathbb{R} &\to [0,1] \\ x &\mapsto \begin{cases} d_j & \text{if } x = x_j, \quad 1 \leq j \leq m, \\ 0 & \text{if } x \notin \{x_1, \ldots, x_m\}. \end{cases} \end{aligned}$$

For this given random variable X, we also define the *cumulative probability distribution function (cdf)* $F(x)$ as the sum of probabilities associated to the x_j which are equal or smaller than x (i.e. $\mathbb{P}\{X \leq x\}$). More formally,

$$\begin{aligned} \text{cdf}_X : \mathbb{R} &\to [0,1] \\ x &\mapsto \begin{cases} 0 & \text{if } x < x_1, \\ d_1 & \text{if } x_1 \leq x < x_2, \\ d_1 + d_2 & \text{if } x_2 \leq x < x_3, \\ \vdots & \vdots \\ 1 & \text{if } x_m \leq x. \end{cases} \end{aligned}$$

[1] https://en.wikipedia.org/wiki/Random_variable

Now, suppose that X is a discrete random variable with its probability mass function depending on a parameter θ. Then, the function

$$\mathcal{L}(\theta|x) = \text{pmf}_\theta(x),$$

considered as a function of θ, is known as the *Likelihood function*.

Discrete random variables are exactly those with a step function *cdf*, namely F, where its discontinuities are finite (or countable) jumps.
When F, the *cdf* of a random variable X, is the integral of another function f (i.e. $f = F'$),

$$F(x) = \int_{-\infty}^{x} f(t)dt, \quad x \in \mathbb{R},$$

then we say that f is the *probability density function (pdf)* of the absolutely continuous random variable X[2]. Similarly, if X is an absolutely continuous random variable, its *Likelihood function* is defined as

$$\mathcal{L}(\theta|x) = f_\theta(x).$$

One way of understanding the difference between frequentists and bayesians is by looking at the *Bayes' billiard problem* (1763). The statement that Bayes wrote is the following:

> Alice and Bob are playing a game in which the first person to get 6 points wins. The way each point is decided is a little strange. The Casino has a pool table that Alice and Bob can't see. Before the game begins, the Casino rolls an initial ball onto the table, which comes to rest at a completely random position, which the Casino marks. Then, each point is decided by the Casino rolling another ball onto the table randomly. If it comes to rest to the left of the initial mark, Alice wins the point; to the right of the mark, Bob wins the point. The Casino reveals nothing to Alice and Bob except who won each point.
>
> Clearly, the probability that Alice wins a point is the fraction of the table to the left of the mark—call this probability $p \in [0, 1]$; and Bob's probability of winning a point is $1 - p$. Because the Casino rolled the initial ball to a random position, before any points were decided every value of p was equally probable. The mark is only set once per game, so p is the same for every point. Imagine Alice is already winning 5 points to 3, and now she bets Bob that she's going to win. What are fair betting odds for Alice to offer Bob? That is, what is the expected probability that Alice will win?

We are asked calculate $\mathbb{P}(\{\text{Bob wins the game}\})$, which is equivalent to calculate the probability that he will win the next 3 points. We set

$$p = \mathbb{P}(\{\text{Alice gets a point}\}),$$

and we clearly see that $\mathbb{P}(\{\text{Bob wins the game}\}) = (1-p)^3$. So, we need to know p, and here's where we'll see the differences between Frequentist and Bayesian approaches.

[2] All the given definitions imply that, necessarily, $f \geq 0$ and $\int_{-\infty}^{+\infty} f = 1$.

The Frequentist approach is basically to estimate p from the data. Given p, we can compute, by means of the binomial formula[3]:

$$\mathbb{P}(A = 5, B = 3|p) = \binom{8}{5} p^5 (1-p)^3. \tag{2.1}$$

Observe that expression (2.1) (which can be seen as a function of p) is the *Likelihood*. It attains its maximum at $\hat{p}_{ML} = \frac{5}{8}$, which is known as the *Maximum Likelihood estimate* of p. This value is the one that is used to estimate p, and with this one, we get

$$\mathbb{P}_{\text{FREQ}}(\{\text{Bob wins the game}\}) = (1-\hat{p})^3 = \frac{27}{512} \approx 0.0527. \tag{2.2}$$

From the point of view of the Bayesian approach, we want to write down exactly the probability we want to infer, in terms only of the data we know, and directly solve the resulting equation — which forces us to deal explicitly with all mathematical difficulties, additional assumptions and uncertainties that may arise. We accept that p is unknown and we treat it as such, integrating over all possible values that a parameter might assume.

In this particular problem, what we want to know is the expected probability that Bob will win. By definition[4], this is the weighted average of $(1-p)^3$ over all possible values of p.

$$\mathbb{P}_{BAYES}(\{\text{Bob wins the game}\}) = \int_0^1 (1-p)^3 \mathbb{P}(p|A = 5, B = 3) dp. \tag{2.3}$$

The problem here is that we need to determine the expression of $\mathbb{P}(p|A = 5, B = 3)$. It can be calculated by Bayes' Theorem[5] and by the Law of Total Probabilities:

Theorem 2.1.1. ***(Bayes' Theorem)*** *Let A and B be two events such that $\mathbb{P}(B) > 0$, then:*

$$\mathbb{P}(A|B) = \frac{\mathbb{P}(B|A)\mathbb{P}(A)}{\mathbb{P}(B)}.$$

Theorem 2.1.2. ***(Law of total probability)*** *Let $\Omega = \biguplus_{n=1}^{N} A_n$ be a finite or countably infinite partition of a sample space (i.e. $N \leq \infty$) such that $\mathbb{P}(A_n) > 0 \ \forall n$. Let A be an event, then*

$$\mathbb{P}(A) = \sum_{n=1}^{N} \mathbb{P}(A_n)\mathbb{P}(A|A_n).$$

Remark. *In the case of a continuous random variable, the sum is substituted by a definite integral on the domain where the parameter is in.*

[3] Suppose that we repeat n times a certain experiment in which we only consider the possibilities of success or failure, with respective probabilities p and $q = 1 - p$. The probability of success or failure in each trial is independent from the others. Letting k be the number of successes, then $\mathbb{P}(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$.

[4] https://en.wikipedia.org/wiki/Expected_value

[5] The name of this theorem–which may be one of the most important theorems ever–is devoted to Thomas Bayes (London, England, 1702 - Tunbridge Wells, 1761). Even though, Martyn Hooper (Hooper, 2013) presents the case of Richard Price (1723–1791), an economist who was a friend of US presidents. The essay where Bayes' theorem was stated for the first time was read to the Royal Society on December 23rd, 1763. It was Price, not Bayes, who communicated the essay to the Royal Society; Thomas Bayes had died two years before.

Retaking *Bayes' billiard problem*, we have that

$$\mathbb{P}(p|A=5,B=3) = \frac{\mathbb{P}(A=5,B=3|p)\mathbb{P}(p)}{\mathbb{P}(A=5,B=3)} \tag{2.4}$$

$$= \frac{\mathbb{P}(A=5,B=3|p)\mathbb{P}(p)}{\int_0^1 \mathbb{P}(A=5,B=3|p)\mathbb{P}(p)dp}. \tag{2.5}$$

The term $\mathbb{P}(p)$ is called "prior", which is potentially problematic, because, by definition, it is a probability of p before any data have been observed. Bayesian methods require specifying prior probability distributions, which are often themselves unknown. For this problem, we assume $\mathbb{P}(p)$ is a constant (since p is uniform on $[0,1]$), which cancels out. Substituting (2.1) in (2.5) and then in (2.3), we end up obtaining

$$\mathbb{P}_{BAYES}(\{\text{Bob wins the game}\}) = \frac{\int_0^1 p^5(1-p)^6 dp}{\int_0^1 p^5(1-p)^3 dp}.$$

The result of both integrals can be determined analytically by the Beta function[6] and the Gamma function[7], and the obtained value is $\mathbb{P}_{BAYES}(\{\text{Bob wins the game}\}) = \frac{1}{11} = 0.090909\ldots$ We can observe that both approaches get quite different results.

2.2 Probabilistic Graphical Models

The main objective of this section is to give an overview of the principal concepts of probabilistic graphical models. It is mostly inspired on the great book (Koller and Friedman, 2009), as well as the lecture notes provided by my supervisor (Hernández-González, 2021).

A *Probabilistic Graphical Model (PGM)* is a probabilistic model which uses graphs to express conditional dependencies between random variables. Suppose that we want to represent a joint distribution $\mathbb{P}$ over some set of random variables $X = \{X_1, \ldots, X_n\}$. In the simplest case, where all the variables were binary, in order to specify a joint probability distribution, we would have to determine $2^n - 1$ numbers. Computationally, it is very expensive to manipulate and generally too large to store in memory. Probabilistic Graphical Model will allow us to be more efficient.

One type of PGM (and the one that we will work with) are *Bayesian Networks*. The core of the Bayesian network representation is a directed acyclic graph (DAG) $\mathcal{G}$. A DAG $\mathcal{G}$ is a pair (V, E) where $V = \{1, \ldots, n\}$ represents the set of *vertices* and $E = \{(u, v) : u, v \in V, u \neq v\}$ represent the set of *arcs*. A DAG is characterized for not having directed cycles. Figure 2.1 displays a toy example of a DAG.
Given a node X_j, we denote $\mathbf{pa}(X_j)$ as the set of parents of X_j (i.e. the nodes with edges directed towards X_j). As an example, looking at Figure 2.1, we have that $\mathbf{pa}(D) = \{B, G\}$.

Bayesian Networks represent a joint probability distribution. Nodes are related to random variables, while edges are related to the simplification of the *chain rule*:

Theorem 2.2.1. ***(Chain rule)*** *Let $A_1, \ldots, A_n$ events such that $\mathbb{P}(A_1, \ldots, A_{n-1}) > 0$. Then,*

$$\mathbb{P}(A_1, \ldots, A_n) = \mathbb{P}(A_1)\,\mathbb{P}(A_2|A_1)\,\mathbb{P}(A_3|A_1, A_2)\cdots\mathbb{P}(A_n|A_1, \ldots, A_{n-1}).$$

[6] https://en.wikipedia.org/wiki/Beta_function
[7] https://en.wikipedia.org/wiki/Gamma_function

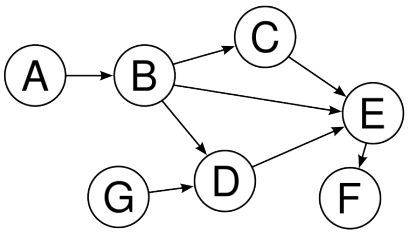


FIGURE 2.1: An example of a directed acyclic graph.

A Bayesian Network can be expressed as a product of conditional probability distributions:

$$\mathbb{P}(X_1, \ldots, X_n) = \prod_{i=1}^{n} \mathbb{P}(X_i | \mathbf{pa}(X_i)).$$

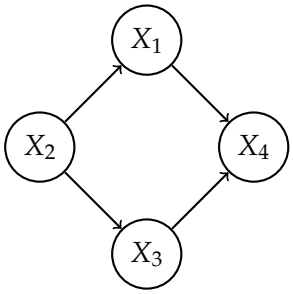


FIGURE 2.2: Toy example of a Bayesian Network.

In the Bayesian Network displayed at Figure 2.2, we can obtain two different factorizations via the chain rule and via Bayesian network's factorization:

- Chain rule:

$$\mathbb{P}(X_2, X_3, X_1, X_4) = \mathbb{P}(X_4 | X_1, X_2, X_3)\, \mathbb{P}(X_1 | X_2, X_3)\, \mathbb{P}(X_3 | X_2)\, \mathbb{P}(X_2).$$

- Bayesian network's factorization:

$$\mathbb{P}(X_1, X_2, X_3, X_4) = \mathbb{P}(X_4 | X_1, X_3)\, \mathbb{P}(X_1 | X_2)\, \mathbb{P}(X_3 | X_2)\, \mathbb{P}(X_2).$$

Clearly, the second one is more efficient to compute. Here, what we have obtained are two conditional independencies. We have seen, from the graph, that X_4 is independent from X_2 given X_1, X_3 and that X_1 is independent from X_3 given X_2.

Given a probability distribution $\mathbb{P}(\mathbf{X}) = \mathbb{P}(X_1, \ldots, X_n)$, a partition of $\mathbf{X} = \mathbf{Y} \cup \mathbf{H} \cup \mathbf{E}$ in three disjoint subsets of variables and an assignment e to the variables in $\mathbf{E}$. The objective of a *conditional probability query* is to find the probability distribution $\mathbb{P}(\mathbf{Y} | \mathbf{E} = e)$. In the general case, it can be rewritten as

$$\mathbb{P}(\mathbf{Y} | \mathbf{E} = e) = \sum_{h} \mathbb{P}(\mathbf{Y}, \mathbf{H} = h | \mathbf{E} = e),$$

but computationally is somehow impossible, since it has exponential complexity. If $\mathbb{P}$ follows a probabilistic graphical model, it can help to reduce complexity, since

many independencies are easier to identify and many wasteful calculations can be avoided. Note that we can represent in this way most of the questions that we usually make to our models in data science.

2.3 Kullback-Leibler divergence

The Kullback-Leibler divergence (Kullback and Leibler, 1951) is a measure for probability distributions over the same random variable X. It is not exactly a distance, since it is not symmetric and it does not verify the triangular inequality. It is denoted as *KL divergence*. Let p and q be two probability distributions, that is, both $p(x)$ and $q(x)$ sum up to 1, and $p(x) > 0$, $q(x) > 0$ for any $x \in X$. Then, the KL divergence is defined as

$$D_{KL}(p,q) = \sum_{x \in X} p(x) \log \left(\frac{p(x)}{q(x)} \right). \tag{2.6}$$

It measures the expected number of extra bits required to code samples from $p(x)$ when using a code based on $q(x)$. Normally, the first distribution (in the case of (2.6), p) is a precisely calculated theoretical distribution, while q represents a model or an approximation of p.

For continuous random variables X, the definition of the KL divergence is extended as

$$D_{KL}(p,q) = \int_{\mathbb{R}} p(x) \log \left(\frac{p(x)}{q(x)} \right) dx. \tag{2.7}$$

In order to see a counterexample which proves that that this measure is not symmetric, we can see Table 2.1, which gives a simplification of one example that is given in (Solomon Kullback, 1959):

TABLE 2.1: Counterexample on the symmetry of the KL divergence

X	$p(x)$	$q(x)$	$p(x) \log \left(\frac{p(x)}{q(x)} \right)$	$q(x) \log \left(\frac{q(x)}{p(x)} \right)$
0	1/6	0.36	−0.12835	0.27724
1	2/3	0.48	0.219	−0.15768
2	1/6	0.16	0.0068	−0.00653
+	1	1	0.09745	0.11303

The KL divergence is always greater or equal than zero, a result known as Gibbs' inequality[8]. It equals zero if and only if $p(x) = q(x)$ almost everywhere.

[8] https://en.wikipedia.org/wiki/Gibbs%27_inequality

Chapter 3

State-of-the-Art

3.1 Introduction to Crowdsourcing

In order to get a first contact with the term "crowdsourcing", we should see (Howe, 2006). This term was coined for the first time in 2005 by Jeff Howe and Mark Robinson in the Wired[1] magazine. They described how companies were using the Internet to "outsource work from the work". With the pass of time, this long concept has been renamed to "crowdsourcing". In fact, crowds have a lot of power, many source software movements (such as Stack Overflow[2] community) have proved that a network of passionate, geeky volunteers can write code just as well as the highly paid developers at Microsoft or Apple. Wikipedia has showed that a crowded model has been able to deliver the most used and efficient encyclopedia of this world.

The first definition of crowdsourcing which Howe came up with was:

> Simply defined, crowdsourcing represents the act of a company or institution taking a function once performed by employees and outsourcing it to an undefined (and generally large) network of people in the form of an open call. This can take the form of peer-production (when the job is performed collaboratively), but is also often undertaken by sole individuals. The crucial prerequisite is the use of the open call format and the large network of potential laborers.

At that moment, the term "crowdsourcing" was only used in the literature. It was not until 2008 that the first research article about this topic was published (Brabham, 2008b). The adaptability of crowdsourcing allows it to be an effective and powerful practice, but makes it difficult to define and categorize. Until that point, the theoretical knowledge base was not solid. In 2012, Enrique Estellés-Arolas and Fernando González Ladrón-de-Guevara provided a wide definition that covered the majority (if not all) of existing crowdsourcing processes until the moment (Estellés-Arolas and L. Guevara, 2012):

> Crowdsourcing is a type of participative online activity in which an individual, an institution, a non-profit organization, or company proposes to a group of individuals of varying knowledge, heterogeneity, and number, via a flexible open call, the voluntary undertaking of a task. The undertaking of the task, of variable complexity and modularity, and in which the crowd should participate bringing their work, money, knowledge and/or experience, always entails mutual benefit. The user will receive the satisfaction of a given type of need, be it economic, social recognition, self-esteem, or the development of individual skills, while

[1] https://www.wired.com/

[2] https://stackoverflow.com/

> the crowdsourcer will obtain and utilize to their advantage that what the user has brought to the venture, whose form will depend on the type of activity undertaken.

In this article, they give an explanation of the three main characteristics that a crowdsourcing model has: (i) who forms the crowd, (ii) what the crowd has to do and (iii) what does the crowd get in return.

3.1.1 Who forms the crowd

The crowd is profiled as a large group of individuals. The magnitude of "large" is not clear, it depends a lot on the problem. In relation to the knowledge of the participants, it can vary too. In general, when a repetitive work is proposed, an expert crowd with special skills is not required. For instance, if we take a look at (Bans, 2010), we can see an example of users that had to give an opinion about some specific products without being experts in those. However, there have been cases where a more expert crowd has been needed. As an example, there exists companies such as Innocentive[3] that propose Research & Development problems whose resolution implies the need of an economic recompense and an expert crowd. In conclusion, the heterogeneity and magnitude of the crowd should depend on the considered problem.

3.1.2 What the crowd has to do

The most important thing that we should take into account is that the task that the crowd is asked to do must have a clear objective. Any non-trivial problem should benefit from crowdsourcing (Doan, Ramakrishnan, and Halevy, 2011). Some authors agree that generic crowdsourcing tasks have to be divisible into lower level tasks, in such a manner that can be accomplished by individual members of the crowd (Heer and Bostock, 2010; Vukovic, Lopez, and Laredo, 2010).

Basically, the crowd has to carry out the resolution of a problem undertaking a task of variable complexity (depends on the problem) and modularity that will imply the voluntary contribution of their work. It is considered that a problem is comprised of any given situation of need held by the initiator of the crowdsourcing activity, e.g., the translation of a fragment of text or opinions about products.

3.1.3 What does the crowd get in return

Some authors (La Vecchia and Cisternino, 2010) state that people in the crowd should be compensated, because they are acting voluntarily. Others (Stewart, Huerta, and Sader, 2009) think that the reward should not be material, but rather a feeling of fulfillment for participating. Many studies (Brabham, 2008a; Brabham, 2010; Lakhani et al., 2007) have analyzed the motivation of the crowd for participating. They highlight some individual needs: the financial reward, the opportunity to develop creative skills, to have fun, to share knowledge, the opportunity to take up freelance work, the love of the community and an addiction to the tasks proposed.

In conclusion, the reward depends on the crowdsourcer, but this one always tries to please the crowd in some way, it can be economic or social.

[3] https://www.innocentive.com/

3.2 Conceptual and mathematical model

In this section we will present an abstract description of our model (conceptual and mathematically), which can be modeled as a probabilistic graphical model. This one is inspired on (Cerquides et al., 2021).

3.2.1 Tasks, workers and annotations

In order to give a real-world context for our model, suppose that a natural disaster has occurred at this moment. It is important to determine which have been the zones that have been more damaged, but due to the chaos that a natural disaster causes, information is scarce. The most recent and instantaneous information we can get comes from social networks, where people who are close to the natural disaster have taken and uploaded images to platforms such as Twitter, Facebook... Collecting data from these social networks is no big deal. Each of the images obtained from the social network can be distributed to a set of citizen scientists that can help labeling that image from a finite set of classes, either as *relevant, no-damage, moderate*... Based on these annotations, we will be able to report to the disaster relief organization.

For our model, we will principally talk about three concepts which have to be defined properly: *task, worker, annotation*:

> **Worker:** Any of the participants in the annotation process. In our example, each of the volunteer citizen scientists involved in labeling images is a worker.
>
> **Task:** The minimal piece of work that can be assigned to a worker. In the previous example, a task would be to label an image.
>
> **Annotation:** The result of the processing of the task by the worker.

3.2.2 Mathematical model

Let $w \in \mathbb{N}$ be the number of workers and $W = \{1, \dots, w\}$ be the set of workers. Let $t \in \mathbb{N}$ be the number of tasks and $T = \{1, \dots, t\}$ the set of tasks. Suppose we have a total of $a \in \mathbb{N}$ annotations and $A = \{1, \dots, a\}$. These are all finite sets, which means that we can define injections to their respective feature space, namely, $\mathcal{W}, \mathcal{T}$ and $\mathcal{A}$. In these spaces we have the characteristics and descriptions associated to each worker, task or annotation. Set $f_W : W \to \mathcal{W}$, $f_T : T \to \mathcal{T}$ and $f_A : A \to \mathcal{A}$. In order to obtain, for each annotation, its associated task and worker, we define $w_A : A \to W$ and $t_A : A \to T$.

We assume that our feature spaces factorize as the cartesian product of some other feature spaces. For the task feature space, we assume that $\mathcal{T} = \mathcal{T}_\mathcal{O} \times \mathcal{T}_\mathcal{C} \times \mathcal{T}_\mathcal{H}$, where $\mathcal{T}_\mathcal{O}$ contains the observable characteristics, $\mathcal{T}_\mathcal{C}$ contains those unobservable characteristics in which we are interested and $\mathcal{T}_\mathcal{H}$ contains those characteristics of the tasks that are unobservable and in which we are not interested. For the workers feature space, we assume $\mathcal{W} = \mathcal{W}_\mathcal{O} \times \mathcal{W}_\mathcal{H}$, where $\mathcal{W}_\mathcal{O}$ contains the observable characteristics and $\mathcal{W}_\mathcal{H}$ contains the unobservable characteristics. We also assume that it exists a space which contains some general characteristics that are relevant for the annotations. We refer to this space $\mathcal{D}$ as the *domain space*.

For our problem, we take as an input:

- w, t and a.
- For each worker $w \in W$ and for each task $t \in T$, its observable characteristics, namely $\mathbf{w}^w_O$, $\mathbf{t}^t_O$.

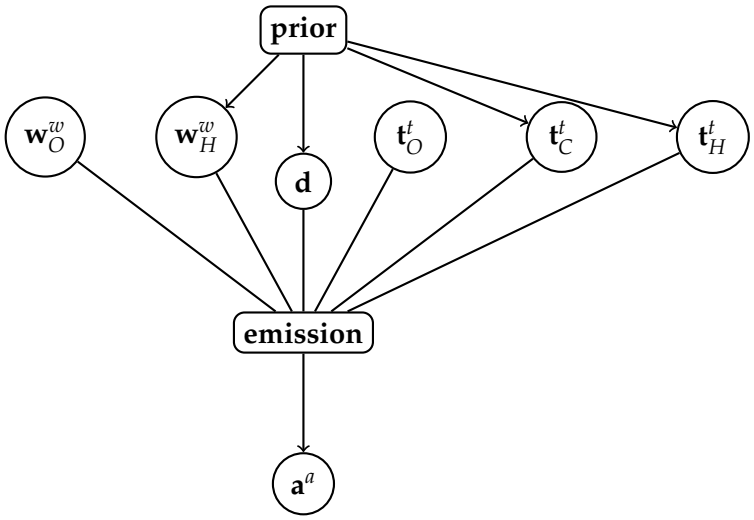


FIGURE 3.1: PGM of the model $\forall t \in T$, $\forall w \in W$ and $\forall a \in A$.

- For each annotation $a \in A$, its corresponding task (i.e. $t_A(a)$), worker (i.e. $w_A(a)$) and annotated characteristics, namely $\mathbf{a}^a = f_A(a)$.
- A probabilistic model of annotation, which consists on:
  - An emission model $\mathbb{P}(\mathbf{a}|\mathbf{w}^w, \mathbf{t}^t, \mathbf{d})$: the probability that, given a certain worker, a certain task and a domain of characteristics $\mathbf{d} \in \mathcal{D}$, the worker annotates the task with label $\mathbf{a}$.
  - A joint prior over every unobservable characteristic

$$\mathbb{P}\left(\mathbf{w}_H^1, \ldots, \mathbf{w}_H^w, \mathbf{d}, \mathbf{t}_C^1, \ldots, \mathbf{t}_C^t, \mathbf{t}_H^1, \ldots, \mathbf{t}_H^t\right).$$

The previous inputs give us the Bayesian Network representation that is displayed in Figure 3.1. This implies that the probability distribution factorizes as

$$\mathbb{P}\left(\mathbf{w}_H^1, \ldots, \mathbf{w}_H^w, \mathbf{d}, \mathbf{t}_C^1, \ldots, \mathbf{t}_C^t, \mathbf{t}_H^1, \ldots, \mathbf{t}_H^t, \mathbf{a}^1, \ldots, \mathbf{a}^a\right) =$$
$$\mathbb{P}\left(\mathbf{w}_H^1, \ldots, \mathbf{w}_H^w, \mathbf{d}, \mathbf{t}_C^1, \ldots, \mathbf{t}_C^t, \mathbf{t}_H^1, \ldots, \mathbf{t}_H^t\right) \prod_{a=1}^{\mathrm{a}} \mathbb{P}\left(\mathbf{a} \middle| \mathbf{w}^{w_A(a)}, \mathbf{t}^{t_A(a)}, \mathbf{d}\right).$$

The objective of this model is to be able to answer probabilistic queries such as $\mathbb{P}(\mathbf{t}_C^1, \ldots, \mathbf{t}_C^t)$ by marginalizing out the rest of the variables. Observe that our model assumes that annotations are independent from each other provided that we are given all the characteristics of the task, the domain and the worker.

3.2.3 Discrete annotation models

In order to particularize our abstract model described in Subsection 3.2.2, we will assume that the feature space of annotations is finite. That is, $\mathcal{A} = \{a_1, \ldots, a_k\}$ with $k < \infty$, and now our model turns out to be a *discrete annotation model*. In this kind of models, each task is considered to have an unobservable characteristic, its "real" label. In other words, $\mathcal{T}_C = \mathcal{A}$. We will work with two different types of discrete annotation models: The *pooled multinomial model* and the *Dawid-Skene models*.

The pooled multinomial model

The pooled multinomial model (Passonneau and Carpenter, 2014; Paun et al., 2018) assumes that all annotators share the same ability. Hence, $\mathcal{W} = \emptyset$. Also, we assume that we can only distinguish a task by its real class. This means that $\mathcal{T}_{\mathcal{O}} = \mathcal{T}_{\mathcal{H}} = \{\emptyset\}$. In short, $\mathcal{T} = \mathcal{T}_{\mathcal{C}} = \mathcal{A}$.

Regarding the domain space, it is the set of probability distributions over $\mathcal{A}$, where $|\mathcal{A}| = k$. This domain can be encoded as a stochastic vector, namely $\tau \in [0,1]^k$, where τ_i denotes the probability of task being of class $\mathbf{a}_i$. As stated before, all the workers share the same behavior, and this "behavior" can be encoded as a stochastic matrix $\pi \in [0,1]^{k \times k}$. Here, $\pi_{i,j}$ denotes the probability of a worker to label a task from real class $\mathbf{a}_i$ with label $\mathbf{a}_j$[4]. In this case, the emission model is

$$\mathbb{P}\left(\mathbf{a}_j \middle| \mathbf{w}, \mathbf{d}, \mathbf{t}\right) \overset{\mathcal{W}=\emptyset \text{ and } \mathcal{T}=\mathcal{A}}{=} \mathbb{P}\left(\mathbf{a}_j \middle| \mathbf{d} = \langle \tau, \pi \rangle, \mathbf{t}_C^l = \mathbf{a}_i\right) = \pi_{i,j}.$$

We assume that the prior for τ and the rows of π is a Dirichlet of vector $\mathbf{1}^k$ (here, $\mathbf{1}^k$ denotes a vector of k ones). Now, the joint posterior factorizes as

$$\mathbb{P}(\mathbf{w}_H, \mathbf{d}, \mathbf{t}_C) \overset{\mathcal{W}=\emptyset}{=} \mathbb{P}(\mathbf{d}, \mathbf{t}_C) = \mathbb{P}(\mathbf{d}) \prod_{l=1}^{\mathsf{t}} \mathbb{P}\left(\mathbf{t}_C^l \middle| \mathbf{d}\right) = \mathbb{P}(\tau)\mathbb{P}(\pi) \prod_{l=1}^{\mathsf{t}} \mathbb{P}\left(\mathbf{t}_C^l \middle| \tau\right), \quad (3.1)$$

where $\mathbb{P}(\tau) = \text{Dirichlet}\left(\tau; \mathbf{1}^k\right)$, $\mathbb{P}(\pi) = \prod_{i=1}^k \text{Dirichlet}\left(\pi_i; \mathbf{1}^k\right)$ and $\mathbb{P}\left(\mathbf{t}_C^l = \mathbf{a}_i \middle| \tau\right) = \tau_i$.

Remark. *The notation "π_i" means "all the i-th row from π", and it is in fact a probability distribution over the true class i.*

The Dirichlet distribution (Lin, 2016) is a multivariate generalization of the Beta distribution. Let $\mathbf{y}^k$ be a vector with k components, where $y_i > 0$ for all i and $\sum_{i=1}^k y_i = 1$. Also, let $\alpha^k = (\alpha_1, \ldots, \alpha_k)$ be another vector of k components such that $\alpha_i > 0$ for all i. Then, the Dirichlet probability density function is given by

$$\text{pdf}\left(\mathbf{y}^k\right) = \frac{\Gamma(\alpha_0)}{\prod_{i=1}^k \Gamma(\alpha_i)} \prod_{i=1}^k y_i^{\alpha_i - 1},$$

where Γ denotes the Gamma function and $\alpha_0 = \sum_{i=1}^k \alpha_i$. Following this definition, it is denoted as Dirichlet $\left(\alpha^k\right)$. Intuitively, we can think α as the "concentration parameter". This one denotes how concentrated the probability mass of the samples are likely to be.

Unpooled models: the Dawid-Skene models

For unpooled models, we distinguish between workers. That is, we assume that different workers might have a different behavior. The three models that we will cover are widely used to model the quality of the workers (Li, 2015). These are: *General Dawid-Skene*, *Class-conditional Dawid-Skene* and *Homogeneous Dawid-Skene*. The first one was originally proposed in (Dawid and Skene, 1979). These are similar to the multinomial model, but they are more general. We firstly provide an abstract description of these models, and then we will adapt them to our framework.

[4]Observe that if π was the identity matrix, then it would mean that our workers are perfect reporters.

- **General Dawid-Skene model:** In this model, each worker w has its own confusion matrix $\pi^w \in [0,1]^{k\times k}$. Formally, we define $\bar{A} = A \cup \{0\}$, where 0 represents the label is missing. We introduce the matrix $Z \in \bar{A}^{w\times t}$, where $Z_{i,j}$ is the label given by the i-th worker to the j-th item, and it will be 0 if the i-th worker did not label the j-th item. We also introduce the indicator matrix $T \in \{0,1\}^{w\times t}$, where $T_{i,j} = 1$ indicates that entry (i,j) is observed, and $T_{i,j} = 0$ that (i,j) is unobserved. Each component of the matrix π^w denotes, for the j-th task:

$$\pi^w_{r,l} = \mathbb{P}\left(Z_{w,j} = l \middle| \mathbf{t}^j_C = r, T_{w,j} = 1\right).$$

  Observe that, for each worker, the number of free parameters to model are $k(k-1)$. It is flexible, but often leads to overfitting on small datasets. The other two models are special cases of the general one, and they consist on imposing constraints on the worker confusion matrices.

- **Class-Conditional Dawid-Skene model:** In this model, the error probabilities of labeling an item with true label i as label j mistakenly for each worker are the same across all $j \neq i$. Formally, for the j-th task:

$$\begin{cases} \pi^w_{r,r} = \mathbb{P}\left(Z_{w,j} = r \middle| \mathbf{t}^j_C = r, T_{w,j} = 1\right), & \forall r \in \bar{A}, \forall w \in W \\ \pi^w_{r,l} = \frac{1-\pi^w_{r,r}}{k-1}, & \forall r, l \in \bar{A} \text{ s.t. } l \neq r, \forall w \in W. \end{cases}$$

  Observe that the off-diagonal elements of each row of the confusion matrix π^w will be the same. Now, the number of free parameters to model for each worker is k.

- **Homogeneous Dawid-Skene model:** Each worker is assumed to have the same accuracy on each class of items, and have the same error probabilities as well. Formally, if the w-th worker labels correctly with accuracy $w_i \in [0,1]$ then, for the j-th task:

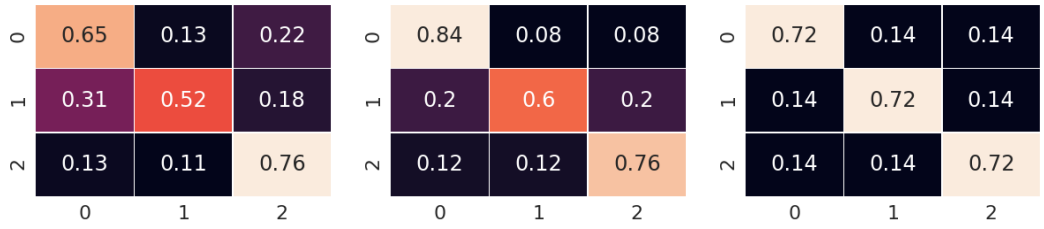
$$\begin{cases} \pi^w_{r,r} = w_i, & \forall r \in \bar{A}, \forall w \in W \\ \pi^w_{r,l} = \frac{1-w_i}{k-1}, & \forall r, l \in \bar{A} \text{ s.t. } r \neq l, \forall w \in W. \end{cases}$$

  In this case, the worker labels an item with the same accuracy, independent of which label this item actually is. Now, the number of free parameters to model for each worker is 1.

Figure 3.2 displays an example of a confusion matrix π for each of the three models. These particular matrices have been generated in the code from Section 5.2. Vertical axis represent the true classes, while horizontal axis represent the predicted classes.

Now that we have a general vision of the models, we can adapt them to our framework. As before, we assume that we can only distinguish a task by its real class (which means that $\mathcal{T}_C = \mathcal{A}$). Now, the general domain characteristics store only vector τ. As stated, each worker w has its own $k \times k$ confusion matrix π^w. In this case, the emission model is

$$\mathbb{P}\left(\mathbf{a}_j \middle| \mathbf{w}, \mathbf{t}, \mathbf{d}\right) = \mathbb{P}\left(\mathbf{a}_j \middle| \mathbf{w}, \mathbf{t}\right) = \mathbb{P}\left(\mathbf{a}_j \middle| \mathbf{w}_H = \pi^w, \mathbf{t}^l_C = \mathbf{a}_i\right) = \pi^w_{i,j}.$$



(a) General (b) Class-Conditional (c) Homogeneous

FIGURE 3.2: Example of a 3×3 confusion matrix for some worker for each of the three Dawid-Skene models.

Regarding the prior, as before, it is assumed to be a Dirichlet of vector $\mathbf{1}^k$ both for τ and for all the rows of the matrices π:

$$\mathbb{P}\left(\mathbf{w}_H, \mathbf{d}, \mathbf{t}_C, \mathbf{t}_H\right) = \mathbb{P}\left(\mathbf{w}_H\right)\mathbb{P}\left(\mathbf{d}\right)\mathbb{P}\left(\mathbf{t}_C \middle| \mathbf{d}\right) = \mathbb{P}\left(\tau\right)\prod_{w=1}^{\mathrm{w}}\mathbb{P}\left(\pi^w\right)\prod_{l=1}^{\mathrm{t}}\mathbb{P}\left(\mathbf{t}_C^l \middle| \tau\right), \quad (3.2)$$

where $\mathbb{P}\left(\tau\right) = \text{Dirichlet}\left(\tau; \mathbf{1}^k\right), \mathbb{P}\left(\mathbf{t}_C^l = \mathbf{a}_i \middle| \tau\right) = \tau_i$ and, analogously to the pooled multinomial model, $\mathbb{P}\left(\pi^w\right) = \prod_{i=1}^k \text{Dirichlet}\left(\pi_i^w; \mathbf{1}^k\right)$.

3.3 Introduction to Stan

Stan (Carpenter et al., 2019) is a programming language that has many utilities: coding probability models, inference algorithms for fitting models and making predictions, posterior analysis tools for evaluating the results... It is written in C++ (Bjarne, 2013). The Stan language is mostly used to specify a (Bayesian) statistical model with an imperative program calculating the logarithm of the probability density function.

3.3.1 Default optimization method in STAN: L-BFGS

STAN provides optimization algorithms that find modes of the specified density. Such modes can be used as parameters estimates or as the basis of approximations to a Bayesian posterior. In fact, this is what we will use in Chapter 5.

The default optimization method in STAN is the *Limited-Memory Broyden Fletcher Goldfarb Shanno algorithm* (L-BFGS) (Nocedal and Wright, 2006; *Stan Reference Manual*). It is an iterative algorithm that requires to be specified a maximum number of iterations; by default, this number is 2000. Convergence is controlled by tolerance values ϵ. The parameters θ_i in iteration i are considered to have converged with respect to tolerance ϵ if

$$\|\theta_i - \theta_{i-1}\| \leq \epsilon.$$

The unnormalized log density $\log \mathbb{P}\left(\theta_i|y\right)$ given some data y is considered to have converged with respect to tolerance δ if

$$\left|\log \mathbb{P}\left(\theta_i|y\right) - \log \mathbb{P}\left(\theta_{i-1}|y\right)\right| < \delta,$$

and it is considered to have converged to within relative tolerance η if

$$\frac{\left|\log \mathbb{P}\left(\theta_i|y\right) - \log \mathbb{P}\left(\theta_{i-1}|y\right)\right|}{\max\left(\left|\log \mathbb{P}\left(\theta_i|y\right)\right|, \left|\log \mathbb{P}\left(\theta_{i-1}|y\right)\right|, 1\right)} < \eta \cdot \varepsilon,$$

where ε is the machine precision. The gradient is considered to have converged to 0 relatively to a specified tolerance γ if $\|g_i\| < \gamma$, where $g_i = \nabla_\theta \log \mathbb{P}\left(\theta^{(i)}|y\right)$ is the gradient at iteration i evaluated at $\theta^{(i)}$.

The algorithm's target problem is to minimize $f(\theta) = \log \mathbb{P}(\theta|y)$. Given an initial point $\mathbf{x}_0$, the algorithm proceeds to determine $\mathbf{x}_1, \mathbf{x}_2 \ldots$ The derivatives of g_i are used as a key driver of the algorithm to identify the direction of steepest descent, and also to find an estimate of the hessian matrix of $f(\theta)$. We take $\mathbf{x}_k$ as the position at the k-th iteration and $g_k = \nabla_\theta f_k = \nabla_\theta \log \mathbb{P}(\mathbf{x}_k|y)$. We assume that we have stored the last updates m of the form

$$s_k = \mathbf{x}_{k+1} - \mathbf{x}_k, \qquad y_k = g_{k+1} - g_k.$$

Normally, m is taken between 3 and 20. The name of this algorithm is devoted to this storage of m updates. We define $\rho_k = \dfrac{1}{y_k^T s_k}$ and choose H_k^0 an approximation of the inverse Hessian at step k. Each step of the method has the form

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k H_k \nabla f_k,$$

where α_k is the step length and H_k is updated as:

$$H_{k+1} = V_k^T H_k V_k + \rho_k s_k s_k^T,$$

where $V_k = Id - \rho_k y_k s_k^T$. In practice, a good method for choosing H_k^0 is to set $H_k^0 = \omega_k Id$, where

$$\omega_k = \frac{s_{k-1}^T y_{k-1}}{y_{k-1}^T y_{k-1}}. \tag{3.3}$$

The limited-memory BFGS algorithm can be stated formally as in Algorithm 1. This optimization algorithm, like most, has difficulties when our function f is not convex.

Algorithm 1 : L-BFGS

Require: $\mathbf{x}_0$, m .
  $k \leftarrow 0$.
  repeat
    Choose H_k^0, for instance as in (3.3).
    Compute $p_k \leftarrow -H_k \nabla f_k$.
    Compute $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \alpha_k p_k$.
    if $k > m$
      Discard the vector pair $\{s_{k-m}, y_{k-m}\}$ from storage.
    Compute and save $s_k \leftarrow \mathbf{x}_{k+1} - \mathbf{x}_k$, $y_k = \nabla f_{k+1} - \nabla f_k$.
    $k \leftarrow k + 1$
  until convergence

3.3.2 Discrete parameters in STAN

A fundamental limitation in STAN is the lack of support for discrete parameters. The difficulty stems from the fact that the construction of Hamiltonian Monte Carlo proposals (which is a Markov chain Monte Carlo sampling algorithm used in STAN)

relies on a numerical solution of a differential equation: approximating a discrete parameter of a likelihood by a continuous differentiable target distribution is difficult in general (Berger, Bernardo, and Sun, 2012). However, Hamiltonian samplers can sample from such models, but it takes more work in thinking about the target distribution. As described in (McElreath, 2018), a general approach to face this issue is the following:

1. Write the probability of an outcome y_i conditional on known values of the discrete parameters. We denote as $\mathcal{L}$ to this conditional likelihood.

2. List all possible states the discrete parameters can take.

   Example. *If we have two binary parameters A, B, there are 4 possible states: $\{0,0\}$, $\{0,1\}$, $\{1,0\}$, $\{1,1\}$. Let $j \in \mathbb{N}$ be the index for each state. Then $j \in \{1,2,3,4\}$.*

3. For each state in (2), compute its probability, which might be provided by our model. Call each state's probability P_j.

4. For each state in (2), compute the probability of an outcome y_i when the discrete parameters take on those values. Use the obtained expression in step 1, inserting the values of the parameters for each state. Denote as $\mathcal{L}_j$ each state's corresponding likelihood.

5. We can compute the unconditional probability of y_i by multiplying each P_j by $\mathcal{L}_j$. Let $M := \sum_j P_j \mathcal{L}_j$ be the sum of these products for all states. This M is in fact, the marginal likelihood: the probability of y_i averaging over the unknown values of the discrete parameters.

Due to numerical stability reasons, we must compute all this steps in the log-probability scale. In this scale, each product $P_j \mathcal{L}_j$ is computed as a sum of logarithms $\log(P_j) + \log(\mathcal{L}_j)$. Applying the essential mathematical property for positive numbers $a = e^{\log(a)}$ and $\log(ab) = \log(a) + \log(b)$, we can compute $\log(M)$ as

$$\log\left(\sum_j e^{\log(P_j)+\log(\mathcal{L}_j)}\right).$$

Chapter 4

Label-switching and a correction proposal

4.1 Label-Switching

When we have to computationally deal with a problem of label aggregation, sometimes, labels switch between them when they should not. In this section, we introduce this challenging problem for inference platforms, and the one that we will face in posterior sections: label-switching (Rodríguez and Walker, 2014; Papastamoulis, 2016).

Let k be a fixed integer which is greater than one, $\mathbf{x} = (x_1, \ldots, x_n)$ a sample of n observations and $\mathbf{z} = (z_1, \ldots, z_n)$ a sequence of unobserved (latent) sequence of state variables, with $z_i \in \{1, \ldots, k\}$ for all i. Let $\Theta \subseteq \mathbb{R}^d$, we define the family

$$\mathcal{F}_\Theta := \{f(\cdot|\theta) : \theta \in \Theta\},$$

where $f \in \mathcal{F}_\Theta$ are distributions. Conditionally to z_i, our observations are distributed according to $x_i|(z_i = j, \theta_j) \sim f(\cdot|\theta_j)$ for $j = 1, \ldots, k$ and $i = 1, \ldots, n$. Assume that z_i are independent random variables that follow a multinomial distribution[1] of probabilities $\mathbf{w} = (w_1, \ldots, w_k)$, which satisfies $w_1, \ldots, w_k > 0$ and $\sum_{i=1}^k w_i = 1$. That is, $\mathbb{P}(z_i = j|w_j) = w_j$ for $j = 1, \ldots, k$. The marginal distribution of x_i is a finite mixture of k distributions:

$$x_i|\theta, \mathbf{w} \sim \sum_{i=1}^{k} w_k f\left(x_i\middle|\theta_k\right). \tag{4.1}$$

If we denote by $p_{t,j}$ to the conditional probability for observation $t \in \{1, \ldots, n\}$ to belong to component $j \in \{1, \ldots, k\}$, applying Definition 2.1.1 and Theorem 2.1.2, we obtain that

$$p_{t,j} = \frac{w_j f\left(x_t|\theta_j\right)}{\sum_{i=1}^k w_i f\left(x_t|\theta_i\right)}. \tag{4.2}$$

[1] The multinomial distribution is a generalization of the binomial distribution. It takes as parameters the number of trials n and the probabilities of each of the k events $p_1, \ldots, p_k$, which must satisfy $\sum_{i=1}^k p_i = 1$. Its support is given by $y_i \in \{0, \ldots, n\}$ for $i = 1, \ldots, k$, with $\sum_{i=1}^k y_i = n$. As a probability mass function, it has

$$\text{pmf}_{multinomial} = \frac{n!}{y_1! \cdots y_k!} p_1^{y_1} \cdots p_k^{y_k}.$$

In this case, the likelihood function is

$$\mathcal{L}\left(\theta, \mathbf{w} \middle| \mathbf{x}\right) = \prod_{i=1}^{n} \sum_{j=1}^{k} w_j f\left(x_i \middle| \theta_j\right).$$

If we introduce the latent parameters, it is written as

$$\mathcal{L}_c\left(\theta, \mathbf{w} \middle| \mathbf{x}, \mathbf{z}\right) = \prod_{i=1}^{n} w_{z_i} f\left(x_i \middle| \theta_{z_i}\right).$$

Let $\mathcal{S}_k$ be the set of all possible permutations between the elements of the finite set $\{1, \ldots, k\}$[2]. Observe that $|\mathcal{S}_k| = k!$. For any $\sigma = (\sigma(1), \ldots, \sigma(k)) \in \mathcal{S}_k$, we denote $\sigma\theta$ as the application of the permutation to vector θ. In particular, we write $\sigma(\mathbf{w}, \theta) := \left(\left(w_{\sigma(1)}, \ldots, w_{\sigma(k)}\right), \left(\theta_{\sigma(1)}, \ldots, \theta_{\sigma(k)}\right)\right)$. The likelihood of the mixture model is invariant for any permutation of the parameters. In fact, given $\rho, \sigma \in \mathcal{S}_k$,

$$\begin{aligned} \mathcal{L}\left(\rho(\theta, \mathbf{w}) \middle| \mathbf{x}\right) &= \prod_{i=1}^{n} \left(\sum_{j=1}^{k} w_{\rho(j)} f\left(x_i \middle| \theta_{\rho(j)}\right) \right) = \prod_{i=1}^{n} \left(\sum_{j=1}^{k} w_{\sigma(j)} f\left(x_i \middle| \theta_{\sigma(j)}\right) \right) \\ &= \mathcal{L}\left(\sigma(\theta, \mathbf{w}) \middle| \mathbf{x}\right). \end{aligned} \tag{4.3}$$

Considering the prior for the parameters $\mathbb{P}(\theta, \mathbf{w})$, the same property is assumed. This can be done by assuming that $\theta \sim \prod_{i=1}^{k} f(\theta_k)$ for a specific family of distributions $f(\cdot)$ which is common to all states (Marin, Mengersen, and Robert, 2005; Frühwirth-Schnatter, 2001). As we have done in Subsection 3.2.3, a common chosen prior for $\mathbf{w}$ is a non-informative Dirichlet distribution.

Let $\mathbb{P}(\theta, w|x)$ be the posterior distribution of the mixture model. We have that $\mathbb{P}(\theta, w|x) \propto \mathcal{L}(\theta, \mathbf{w}|\mathbf{x})\, \mathbb{P}(\theta, w)$, which means that $\mathbb{P}(\theta, w|x)$ inherits (because of (4.3) and the previous statement) the property of invariance under permutations. All the marginal densities of component specific parameters θ and weights $\mathbf{w}$ will coincide. Hence, in a Markov chain Monte Carlo algorithm (MCMC) (Walsh, 2004), the indices of the parameters can permute multiple times between iterations. There exists the high risk that the output from the MCMC sampler converges to a symmetric posterior distribution, which will cause the generated values to switch between the symmetric high posterior density areas.

When we have to deal with a label aggregation problem, label-switching is commonly an issue. It turns out that the optimization/sampling algorithm that we use returns us an output with switched indices of θ. In other words, two or more of the inferred labels are exchanged.

Example. *Suppose that our generative distribution is given by* $\tau = (0.2, 0.3, 0.5)$. *We say that we are facing a label-switching problem when the inferred parameters (by means of an optimization/sampling algorithm) returns us values such as* $\hat{\tau} = (0.5, 0.2, 0.3)$, $\hat{\tau} = (0.3, 0.2, 0.5) \ldots$

4.2 Proposal of a correction algorithm

In order to propose a solution/correction to the label-switching problem, we must assume some hypothesis on our workers:

[2] https://en.wikipedia.org/wiki/Symmetric_group

- Workers are "good" annotators. This assumption means that, if a worker sees a task of class k, most of the times will label it as class k. In terms of matrices π, this means that our squared matrices π will be mostly diagonal dominant[3].

Given this assumptions, our proposal to correct the label-switching problem can be understood with the following colloquial sentence: "each row of π wants to have its maximum at the diagonal". We would like to correct label-switching without any previous intervention to the optimization/sampling method. The pipeline is the following:

1. Given some data, we apply General Dawid-Skene (3.2.3) and we get specific values for τ and π, which might (or not) have been label-switched.
2. For each matrix π^w, we search where is the maximum of each row and we store the resulting array, which contains the index of every row's maximum.
3. We select the most repeated resulting array, which we will call "permutation". Sometimes, this resulting permutation is modified, we will later see more details about that.
4. Apply the transformation indicated by the permutation to all the matrices π and also to vector τ.

Example. *In order to understand the correction, let's analyze the following toy example. Suppose that we have three workers and that STAN infers the following confusion matrices for them:*

$$\pi^1 = \begin{pmatrix} 0.2 & 0.3 & 0.5 \\ 0.38 & 0.56 & 0.06 \\ 0.8 & 0.1 & 0.1 \end{pmatrix}, \pi^2 = \begin{pmatrix} 0.1 & 0.2 & 0.7 \\ 0.28 & 0.66 & 0.06 \\ 0.75 & 0.15 & 0.1 \end{pmatrix}, \pi^3 = \begin{pmatrix} 0.15 & 0.25 & 0.6 \\ 0.2 & 0.6 & 0.2 \\ 0.7 & 0.2 & 0.1 \end{pmatrix}.$$

Our resulting permutation to apply for this case word be $\sigma = (3,2,1)$. That is, first row should be the third, second row is correct and third row should be the first one.

This example is an ideal case, since we can clearly see for each row which might be its truly position. In other words, σ contains three different numbers. However, in some cases, the resulting permutation may require a slight modification (i.e. step 3). Unfortunately, this permutation will not always contain unique values. That is, if we have k classes (i.e. our matrices are of size $k \times k$), our permutation array (which will be of size k) will not always contain all the numbers from 1 to k. See the following example:

Example. *Suppose that, unfortunately, the majority behavior is inferred to be associated to the following confusion matrix (which has been label-switched):*

$$\begin{pmatrix} 0.8 & 0.1 & 0.1 \\ 0.2 & 0.3 & 0.5 \\ 0.48 & 0.46 & 0.06 \end{pmatrix}.$$

Our resulting permutation would be $\sigma = (1,3,1)$. Two different rows want to be the first one. Clearly, we need to slightly modify σ.

[3]Purely mathematically speaking, the term "squared diagonal matrix" is not the most adequate here. A squared matrix $A = (a_{i,j})_{i,j}$ is said to be diagonal dominant if $|a_{i,i}| \geq \sum_{j \neq i} |a_{i,j}|$ for all i. That is, for every row of the matrix, the magnitude of the diagonal entry in a row is larger than or equal to the sum of the magnitudes of all the other (non-diagonal) entries in that row. In our context, we will understand a matrix to be "diagonal dominant" if for each row, the maximum is at the diagonal.

Given a permutation $\sigma \in \{1, \dots, k\}^k$, the modification consists on the following steps:

1. Fix the components that only appear once. For instance, in the previous example, we would fix the second position of σ. With this step, we ensure that rows that clearly have a place to go are not modified.

2. For the rest of components (which are in fact, repeated), fix the ones that its value coincides with its index. In fact, if the j-th row already has its maximum at the diagonal, it should not be modified just because another row has its maximum at the j-th position.

3. The remaining components are filled with the missing values in an ascending way.

If we look at the previous example, permutation $\sigma = (1, 3, 1)$ would be transformed to $\hat{\sigma} = (1, 3, 2)$. The assumption that our workers are mostly non-misleading annotators will prevent us from applying this modification many times. The pseudo-code for this proposal is given in Algorithm 2.

Algorithm 2 : Label-switching correction

Require: $\tau, \pi^1, \ldots, \pi^w$.
  Set $\hat{\tau} \leftarrow \tau$.
  Initialize empty list *permutations*.
  for $i = 1$ to w **do**
    Set $\hat{\pi}^i \leftarrow \pi^i$.
  end for
  for $i = 1$ to w **do**
    maximums $\leftarrow \text{argmax}(\pi^i, axis = 1)$
    Store *maximums* in *permutations*.
  end for
  most_repeated $\leftarrow$ Get most repeated array in *permutations*.
  final_permutation $\leftarrow$ ApplyCorrection(*most_repeated*).
  Apply *final_permutation* to $\hat{\tau}, \hat{\pi}^1, \ldots, \hat{\pi}^w$.
  Return $\hat{\tau}, \hat{\pi}^1, \ldots, \hat{\pi}^w$.

  Function ApplyCorrection(*list*):
  uniques $\leftarrow$ Get values that appear exactly once in *list*.
  if len(*uniques*) == len(*list*) **then**
    Return *list*.
  end if
  existing $\leftarrow$list(set(*list*)).
  rank $\leftarrow$ list(range(len(*list*)))
  missing $\leftarrow$ Get values in *rank* that are not in *existing*.
  Initialize empty list *corrected*.
  for $k = 1$ to len(*list*) **do**
    if *list*[k] not in *uniques* **then**
      if *list*[k] not in *corrected* **then**
        if *list*[*list*[k]] == *list*[k] **then**
          if $k \neq$ *list*[k] **then**
            list[k] $\leftarrow$ *missing*[0].
            missing.pop(0).
          else
            Store *list*[k] in *corrected*.
          end if
        else
          Store *list*[k] in *corrected*.
        end if
      else
        list[k] $\leftarrow$ *missing*[0].
        missing.pop(0).
      end if
    else
      continue
    end if
  end for
  return *list*.

Chapter 5

Empirical analysis

The objective of this chapter is to present the results of the code, as well as to explain the theoretical concepts that they require. All the necessary codes for this chapter can be found in https://github.com/apadros01/TFM-CrowdLearning.

For testing our models, we will use the dataset that is contained in the file `multinomial.json`. If we take a look at the file, we can see that it contains:

- A total of $w = 20$ workers.
- A total of $t = 1000$ tasks.
- A total of $a = 10000$ annotations.
- $k = 3$ different classes.

Each task has been labeled by 10 different workers. As we can seen in Figure 5.1, not all the workers have annotated the same number of tasks, but there are no major differences between the total number of tasks for each one. Regarding the distribution of the classes, we can clearly see in Figure 5.2 that the majority of the annotations are from class 2.

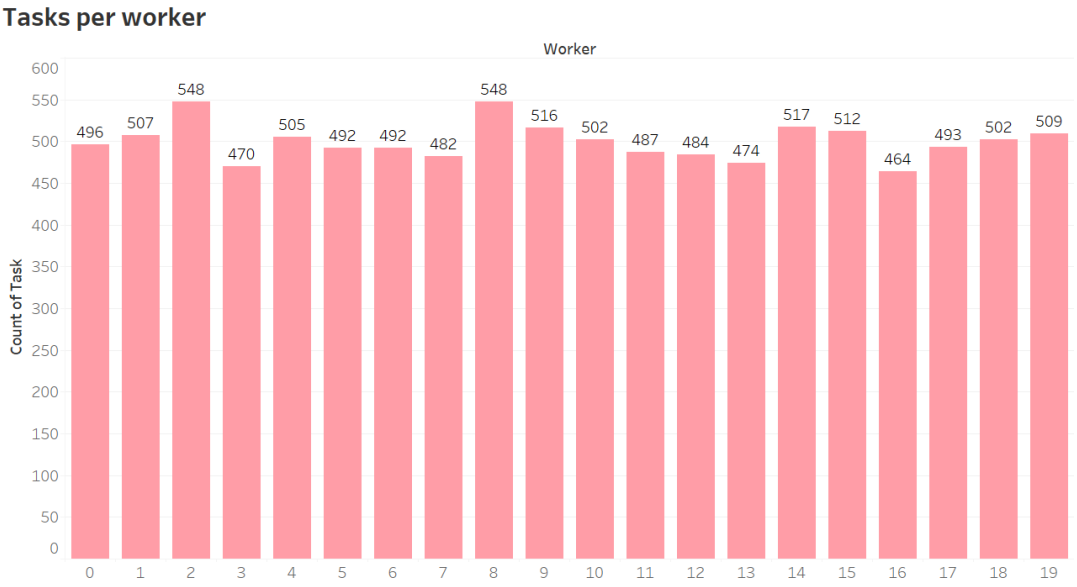


FIGURE 5.1: Number of tasks that each worker has annotated.

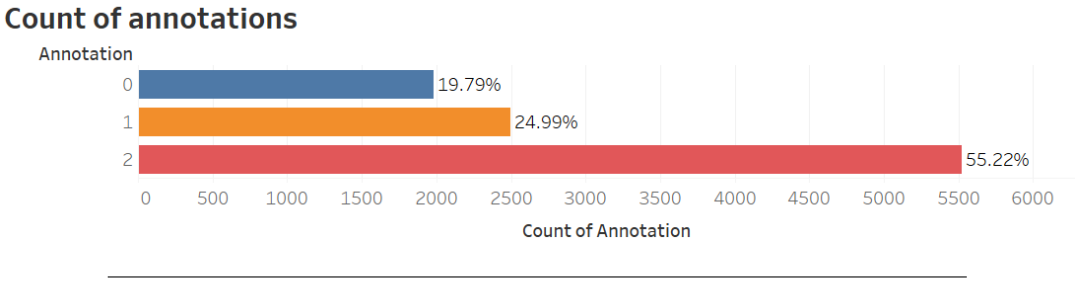


FIGURE 5.2: Number of annotations for each class.

5.1 Pooled multinomial

Our first objective is to determine the best parameters τ and π which have generated our data by the pooled multinomial model. The name of this model is devoted to two factors: (i) the word "pooled", as stated before, indicates that all annotators share the same ability. (ii) The word "multinomial" is due to the assumption that, if we denote as $x_i \in \mathbb{Z}^k_{\geq 0}$ to the vector of accumulated annotations for each of the k classes for the i-th task[1], $\mathbf{t}^i_C$ as the real class of the i-th image, then $x_i \sim \text{multinomial}\left(\pi_{\mathbf{t}^i_C}\right)$.

In this section, we will present two ways to code the pooled multinomial model. For both, observe that the real class $\mathbf{t}_C$ is a discrete parameter, as explained in 3.3.2, it takes some work in thinking about the target distribution, because discrete parameters are not supported in STAN, so we have to marginalize them out. For our first method, let t be the number of tasks, k the number of classes and $x \in \mathbb{Z}^{t\times k}_{\geq 0}$ the matrix of accumulated annotations for each class (each row of x represents a task), then:

$$\mathbb{P}\left(x\middle|\pi,\tau\right) \overset{\text{tasks are indep.}}{=} \prod_{i=1}^{t} \mathbb{P}\left(x_i\middle|\pi,\tau\right) \overset{\text{Theorem 2.1.2}}{=} \prod_{i=1}^{t}\sum_{\mathbf{t}^i_C} \mathbb{P}\left(x_i\middle|\mathbf{t}^i_C,\pi_{\mathbf{t}^i_C}\right)\mathbb{P}\left(\mathbf{t}^i_C\middle|\tau\right). \tag{5.1}$$

Observe that $\mathbb{P}\left(x_i\middle|\mathbf{t}^i_C,\pi_{\mathbf{t}^i_C}\right) = \text{multinomial}\left(x_i;\pi_{\mathbf{t}^i_C}\right)$, and if we substitute in the sum of (5.1), we obtain:

$$\mathbb{P}(x|\tau,\pi) = \prod_{i=1}^{t}\sum_{\mathbf{t}^i_C} \text{multinomial}\left(x_i;\pi_{\mathbf{t}^i_C}\right)\mathbb{P}\left(\mathbf{t}^i_C\middle|\tau\right).$$

As stated in 3.2.3, we assumed that $\mathbb{P}\left(\mathbf{t}^i_C|\tau\right) = \tau_i$, which means that the obtained expression can be treated computationally. Since $\mathbf{t}_C$ is a latent discrete parameter, we must take the logarithm of this expression and, using the essential mathematical properties for positive numbers $e^{\log(a)} = a$ and $\log(ab) = \log(a) + \log(b)$, we end up obtaining

$$\log\left(\mathbb{P}(x|\tau,\pi)\right) = \sum_{i=1}^{t}\log\left(\sum_{\mathbf{t}^i_C}\exp\log\left[\text{multinomial}\left(x_i;\pi_{\mathbf{t}^i_C}\right) + \log\left(\mathbb{P}\left(\mathbf{t}^i_C|\tau\right)\right)\right]\right). \tag{5.2}$$

[1]Example: suppose that we are looking at the 4-th task and that there is a total of $k = 3$ classes. Imagine that we observed that 5 people labeled the image as class 1, 6 people as class 2 and 7 people as class 3. In this case, the vector of accumulated annotations for the 4-th task would be $x_4 = (5,6,7)$.

Expression (5.2) plays a big role in the following code that we will treat. For the code that is included in the STAN file `multinom_confusion_matrix.stan` (whose associated notebook is `pooled_multinomial.ipynb`), we have applied a little trick to avoid the label-switching phenomenon (4.1) –which we will treat more properly in posterior sections. The trick consists on selecting an appropriate initial point for the optimization algorithm that STAN performs. In this case, we have chosen to start from an initial matrix which is near to the identity. The obtained parameters are

$$\begin{aligned} \hat{\tau} &= (0.11447845, 0.29650916, 0.58901239)\,, \\ \hat{\pi} &= \begin{pmatrix} 0.68293363 & 0.10402475 & 0.21304162 \\ 0.21832192 & 0.59436654 & 0.18731154 \\ 0.09335238 & 0.10484805 & 0.80179957 \end{pmatrix} \end{aligned} \tag{5.3}$$

This code works well, but it has a big issue, and it is matrix x. If we would like to go further away, this matrix would not give enough information to us. In other words, it is limited, since it does not tell us anything about the behavior of the workers. For the pooled multinomial model, we do not want to model each worker's confusion matrix, but it is somehow natural to think about doing it in the future. An alternative way (i.e. our second method) to write the code for the pooled multinomial model and also taking into account what each worker has annotated is covered in the STAN file `Multinomial.fit_and_consensus.stan`, whose associated notebook is `pooled_multinomial_optimal.ipynb`. As an input, it takes many of the functions defined in Subsection 3.2.2:

- The number of workers w, the number of classes k, the number of tasks t and the number of annotations a.
- Three arrays of a components: the first one comes from the function t_A, which returns, for each annotation, its associated task. The second one comes from the function w_A, and it contains, for each annotation, its associated worker. Finally, an array with all the annotations for each task and worker.

The analogous marginalization of the discrete parameter for this new way of writing the code would be given by taking the logarithm of Equation 3.1. With this code, we obtain the same results as in (5.3), and it will be easier to generalize it to the Dawid-Skene models.

5.2 Dawid-Skene models and label-switching

In order to have a first impression for the Dawid-Skene models (3.2.3), we should see the notebook `dawid-skene-models.ipynb`, whose associated STAN files are: `general_dawid-skene.stan`, `conditional_dawid-skene`, `homogeni_dawid-skene.stan`. As the names of the STAN files indicate, this notebook contains the three types of the Dawid-Skene models. The marginalization process of the discrete parameter for these codes is given by Equation 3.2. The conditional and homogeneous models are interesting to see, but they will not allow us to deeply study the label-switching problem due to all the constraints that they require. For this reason, we will only work with the general version.

5.2.1 Addition of external classes

Until now, we have considered that all annotations give specific information about the task. For instance, we have been treating the problem as if all the annotators gave

an answer such as *blue*, *red*, *green*, ... In many real life situations, our annotators will not always be able to give an answer, and they will response something like *I'm not sure, I don't know* ... These kinds of answers give information about the true label of the task too, and it is interesting to consider them as well. We call them "external classes" and, analogously, the "valid" answers are called as "internal classes".

Notebook `extern_classes.ipynb` contains the code for this generalization. The big difference with respect to the previous case lies in the size of matrices π. In the previous cases, the matrices π were square, rows represented the true label and columns the predictions by each particular worker. Let $k \in \mathbb{N}$ be the number of internal classes and $m \in \mathbb{N}$ the number of external classes. Until now, matrices π have had size $k \times k$. The main idea for adding external classes is to generalize the codes to work with matrices π of size $k \times (k+m)$. The STAN code with this generalization is included in `extern_classes.stan`. As an example, we have slightly modified the dataset, and we have switched 150 random annotations to an extern class. For instance, the obtained confusion matrix for the third worker has been

$$\hat{\pi}^3 = \begin{pmatrix} 0.605 & 0.086 & 0.294 & 0.016 \\ 0.208 & 0.589 & 0.183 & 0.02 \\ 0.074 & 0.109 & 0.802 & 0.015 \end{pmatrix}.$$

The last column represents, given a true label, the probability that the annotator classified the task with an external class.

5.2.2 Label - switching phenomenon

For this Subsection, we have only considered internal classes and we have worked with General Dawid-Skene, since it is the one which gets more freedom to the outputs and it can be affected by the label-switching problem.

Motivation

When we have applied the STAN optimization method with General Dawid-Skene, we have also applied the same trick than in the pooled multinomial case (5.1), which consisted on starting the default STAN optimization algorithm (3.3.1) from an initial matrix π that is near to the identity. For instance, let's see what has happened to τ and to the 6-th worker whether we apply this trick or not:

- Applying the trick:

$$\hat{\tau} = \begin{pmatrix} 0.11254806 & 0.29708402 & 0.59036792 \end{pmatrix},$$
$$\hat{\pi}^6 = \begin{pmatrix} 0.64342157 & 0.15134315 & 0.20523528 \\ 0.16934618 & 0.6049475 & 0.22570632 \\ 0.09377215 & 0.08709695 & 0.8191309 \end{pmatrix}.$$

- Not applying it:

$$\hat{\tau} = \begin{pmatrix} 0.5903699 & 0.11257069 & 0.29705941 \end{pmatrix},$$
$$\hat{\pi}^6 = \begin{pmatrix} 0.09377598 & 0.08709459 & 0.81912943 \\ 0.6434031 & 0.15136465 & 0.20523226 \\ 0.16929417 & 0.60498277 & 0.22572306 \end{pmatrix}.$$

Clearly, we can observe that it looks like, when we do not apply the trick, the values have been permuted. In particular, the second and third elements of $\hat{\tau}$ should move one place to the left and the first one should be in the last place. The same permutation should be applied to matrix $\hat{\pi}$ rows. We are facing a problem of label-switching.

Evaluation

In order to evaluate our method for label switching correction, we will follow the following pipeline:

1. Given t tasks, w workers, a fixed number of annotations per task, a vector τ and w matrices π. We generate a dataframe that contains, for each task and some workers, an annotation. The STAN file that generates it is `generate_data.stan`.
2. From this dataset, we apply General Dawid-Skene and we learn new values $\hat{\tau}$ and $\hat{\pi}^w$.
3. If necessary, we apply our label-switching algorithm to $\hat{\tau}$ and $\hat{\pi}^w$.
4. We apply the Kullback-Leibler divergence (2.3) to measure the distance from the generative probability distributions given by τ and π to the learned distributions $\hat{\tau}$ and $\hat{\pi}$.

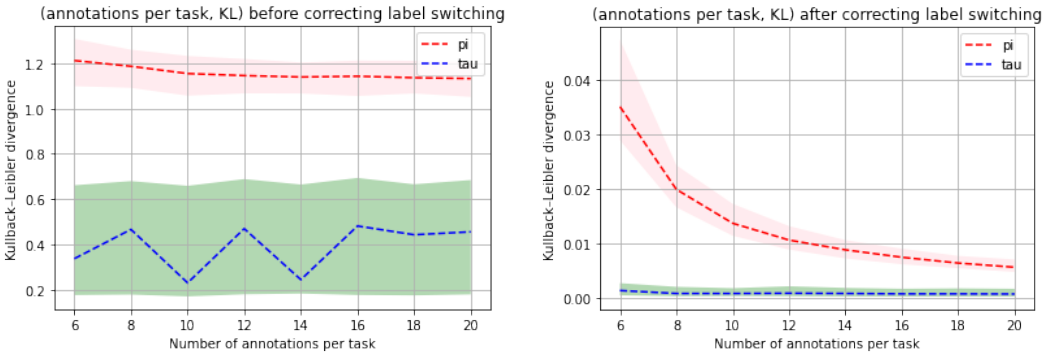
We have studied the Kullback-Leibler values when the number of annotations per task increases. For each number of annotations per task, we have repeated n times the previous pipeline. This value of n, by default, has been set to 500. Regarding the priors, we use $\mathbb{P}(\tau) = \text{Dirichlet}\left(\tau; \mathbf{1}^k\right)$ and $\mathbb{P}(\pi) = \prod_{i=1}^{k} \text{Dirichlet}\left(\pi_i; \mathbf{1}^k\right)$.

Our first evaluation has been done with the dataset that is contained in the file `multinomial.json`. This particular experiment is special, since we do not know the generative distributions of τ and π. In order to obtain them, we have applied General Dawid-Skene to the dataset, starting from an initial matrix which is close to the identity. The results can be seen in Figure 5.3. This experiment (and the next one) can be found in the `generate_data.ipynb` notebook. The obtained lines represent the median of the n experiments, while the shadows represent the 25-th and 75-th quantile. Before applying label-switching, the KL values for τ and π are quite large and they stabilize, which means that the algorithm is not learning. Variances are high too. After applying our label-switching correction, we can see that variances decrease as we increase the number of annotations per task, as well as the median. Table 5.1 shows the mean value for the medians. Clearly, the values are better after applying label-switching.

TABLE 5.1: Mean value of the medians for the JSON file.

	τ	π
Before correcting label-switching	0.390957	1.1564773
After correcting label-switching	0.000912	0.013486

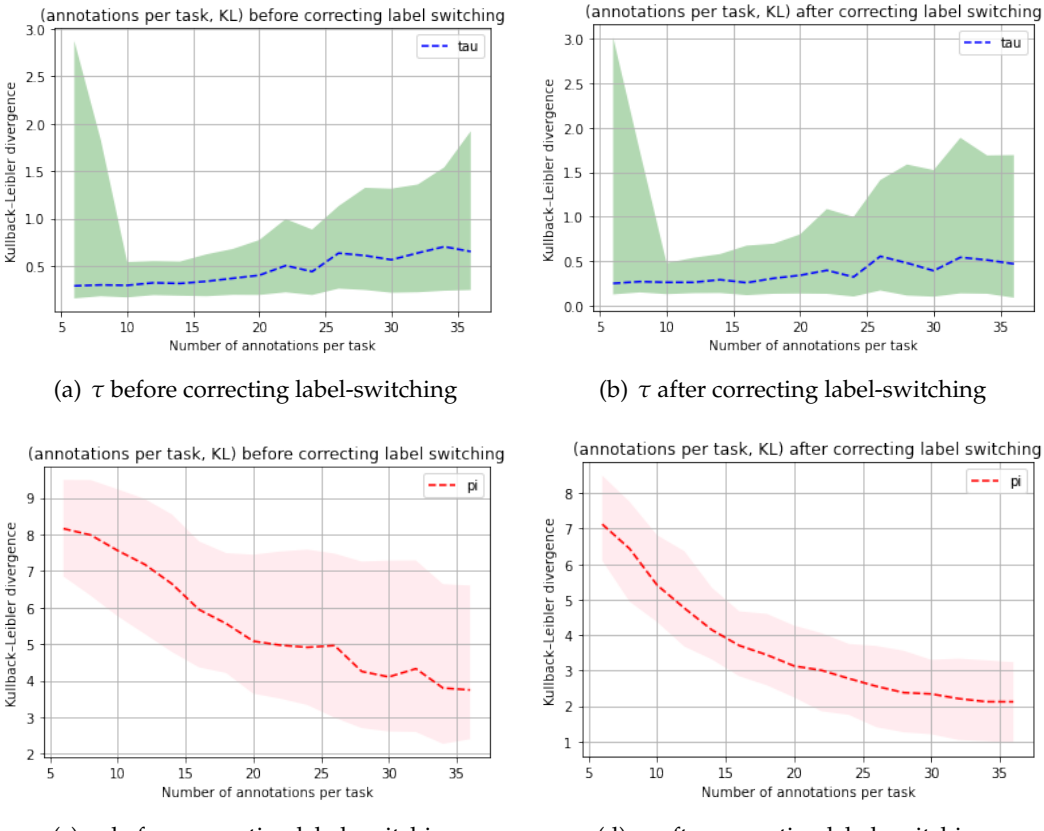
For the following cases, we have worked with synthetic data. It may be interesting to see how the algorithm performs when it is outside the assumptions we had made. For the following experiment, we have created a set of workers that are not good annotators, in general. In terms of matrices, this means that the matrices are not completely diagonal dominant. Due to the general idea of Algorithm 2,



(a) Before correcting label-switching (b) After correcting label-switching

FIGURE 5.3: KL values for the data from multinomial.json file.

this experiment has been made expecting bad results. As a guide, our workers are similar to the ones that have been predicted in Section 5.5 from (Cerquides et al., 2021). For this particular experiment, we have $k = 5$ classes, 50 workers, 800 tasks and a balanced distribution for the true labels, which has been randomly chosen by $\tau \sim \text{Dirichlet}\left(\mathbf{15}^k\right)$. As expected, the results for this case are bad (see Figure 5.4): medians are large and variances too.



(a) τ before correcting label-switching (b) τ after correcting label-switching

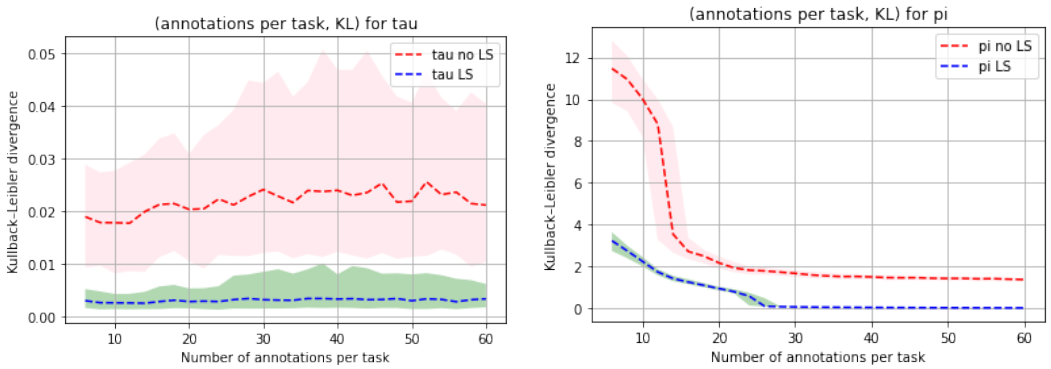
(c) π before correcting label-switching (d) π after correcting label-switching

FIGURE 5.4: KL values for the synthetic dataset of workers with an associated non-diagonal dominant confusion matrix.

In our next experiment, we change the number of classes to $k = 4$. We promote a balanced distribution of the true labels by imposing a prior such as $\tau \sim \text{Dirichlet}\left(\mathbf{30}^k\right)$. Our $w = 100$ workers have now been created to be great annotators: that is, their associated confusion matrices π^w are diagonal dominant. We have set the number of tasks to $t = 500$. Working with artificial data is sometimes problematic. For this reason, we have decided to give a little help to the optimization methods in this experiment (and the following one too), but not enough help to fix the label-switching problem. This experiment and the following one are in the `initial_points_help.ipynb` notebook. This little help consists of, before applying step 2 of the pipeline, we obtain a general $\hat{\pi}$ matrix and a vector $\hat{\tau}$ from the pooled multinomial. We use these two parameters as initial points for the General Dawid-Skene. Additionally, we have given an initial point for τ to the pooled multinomial too. This initial point is given by the annotations, applying *majority voting*. This is a basic aggregation rule and consists of, for each task, we take as the actual label the most voted class. Majority voting can be written formally as

$$\hat{\mathbf{t}}_C^j = \text{argmax}_{k \in [L]} \sum_{i=1}^{w} \mathbb{I}\left(Z_{i,j} = k\right),$$

where $[L]$ denotes the label set, $\mathbb{I}$ is the indicator function and, as we have defined in (3.2.3), $Z_{i,j}$ is the label given by the i-th worker to the j-th item, and it is 0 if this worker did not label the j-th item. Figure 5.5 displays the obtained results. These new plots have a different configuration from the previous ones, now we compare each estimated parameter with its label-switching corrected version in the same plot.



(a) τ before and after applying label-switching. (b) π before and after applying label-switching.

FIGURE 5.5: KL values for the experiment with artificial data, characterized by good annotators, a balanced distribution of the true labels and 4 classes.

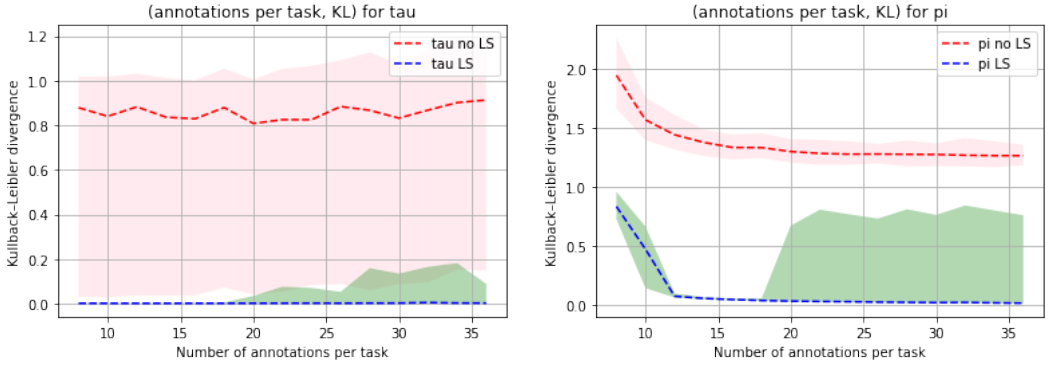
For τ, we can see that the values are in general low, even though the variance before label-switching is high. The reason of these low values for τ is not a surprise, since we have generated a very balanced distribution of the true labels and, even if we do not correct label-switching, all the components of τ are similar. For this experiment, it is interesting to see how well have improved the KL values for π. These specific numbers can be seen in Table 5.2.

Our last experiment consists of the case of an unbalanced τ. That is, a component τ_i which is much larger than the rest. In terms of labels, this means that the distribution of true classes is biased towards a single one. We set $k = 4$ classes, 36 workers,

TABLE 5.2: Mean value of the medians for the artificial data.

	τ	π
Before correcting label-switching	0.0219	2.978354
After correcting label-switching	0.003041	0.615390

1000 tasks and we randomly select $\tau \sim \text{Dirichlet}(75, b_2, b_3, b_4)$, where b_2, b_3, b_4 have been chosen to be random integers between 10 and 15. For computation time reasons, we have reduced the number of repetitions for each experiment to $n = 400$. Figure 5.6 displays the results.



(a) τ before and after correcting label-switching. (b) π before and after correcting label-switching

FIGURE 5.6: KL values for the experiment with a distribution of true labels biased towards a single one.

The medians are in general good, we can clearly see that the KL values after correcting label-switching are low. Table 5.3 for the specific values. The big issue have been the variances. As stated in (3.3.1), *L-BFGS* has important drawbacks when our optimization problem is not convex. It is part of the Quasi-Newton methods family, which are used to either find zeros or **local** maxima and minima of functions. For this reason, this method is quite sensible to the initial point that it starts with. In this particular experiment, our target function has many local optimal points and our parameters are (sometimes) converging to a point that it is not even correctable with label-switching. That is, suppose that our generative distribution is $\bar{\tau} = (0.1, 0.3, 0.6)$. When our given initial point is not good enough, *L-BFGS* might converge to a point such as $(0.47, 0.23, 0.3)$, which can not be corrected by only switching labels. In our particular experiment, since τ is biased towards a single label, many of the "wrong" points where the algorithm converges have a component which is very close to zero, causing KL values to be large (see the formula (2.6)). One way to reduce this variance is to increase the number of repetitions of the experiment and the number of tasks, but this clearly has an expensive computational cost. It could seem that to give an initial point to matrix π in the pooled multinomial optimization may be a way to solve the problem of ending up in an incorrect optimal point. Actually, we have tested this idea and it is clearly effective. However, we have not considered it in this study because we were under the hypothesis that label switching could be corrected without any previous intervention, which has been shown to be non-completely true. This idea is *too* effective, and it does not even provoke label-switching.

TABLE 5.3: Mean value of the medians for the unbalanced case.

	τ	π
Before correcting label-switching	0.857524	1.370039
After correcting label-switching	0.002148	0.120153

5.3 Conclusions and future work

In this work, we have proposed an algorithm that aims to correct the label-switching problem without having to intervene in the optimization/sampling methods of our inference platform. This algorithm (see Algorithm 2) has proven to be very effective when our workers are mostly non-misleading. That is, when our set of workers tends to correctly label tasks. The experiments that are related to Figure 5.3 and 5.5 have shown that. Outside of the hypothesis we have assumed, the algorithm has proven to have difficulties (see Figure 5.4). Unexpectedly, we have run into one of the biggest problems (if not the biggest) in mathematical optimization: non-global optima. When we have tested our algorithm with an unbalanced generative distribution of the true labels, we have observed that STAN optimization methods (such as *L-BFGS* (3.3.1)) infer points that are not correctable with our proposed algorithm. This has led us to obtain unstable results, but still quite good (see Figure 5.6).

As future work, we have thought of proposing an improvement in the function *ApplyCorrection()* from Algorithm 2, but it has not been possible to carry it out due to lack of time. As it is explained in the literature, specifically, in Step 3, the remaining components are filled in an ascending way. Clearly, this step introduces an error, which ends up being minimal when we make lots of experiments. It might be a good idea to modify this "random filling" and do it by taking into account the form of the confusion matrices. In this way, this introduction of error is reduced, although computationally it may be more expensive.

Bibliography

Bans, Lauren (2010). "Using crowdsourcing to control Inventory". In: URL: https://www.inc.com/magazine/20100201/using-crowdsourcing-to-control-inventory.html.

Berger, James O., Jose M. Bernardo, and Dongchu Sun (2012). "Objective Priors for Discrete Parameter Spaces". In: *Journal of the American Statistical Association* 107.498, pp. 636–648. DOI: 10.1080/01621459.2012.682538. eprint: https://doi.org/10.1080/01621459.2012.682538. URL: https://doi.org/10.1080/01621459.2012.682538.

Bjarne, Stroustrup (2013). *The C++ programming language/Bjarne Stroustrup.–*.

Brabham, Daren (June 2008a). "Moving the crowd at iStockphoto: The composition of the crowd and motivations for participation in a crowdsourcing application". In: *First Monday* 13. DOI: 10.5210/fm.v13i6.2159.

— (Dec. 2010). "Moving the crowd at threadless". In: *Information* Communication & Society, pp. 1122–1145. DOI: 10.1080/13691181003624090.

Brabham, Daren C. (2008b). "Crowdsourcing as a Model for Problem Solving. An Introduction and Cases". In: *Convergence* 14.1, pp. 75–90. DOI: 10.1177/1354856507084420. eprint: https://doi.org/10.1177/1354856507084420.

Carpenter, Bob, Andrew Gelman, Matthew D Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell (2019). "Stan: A probabilistic programming language". In: *Journal of statistical software* 76.1. URL: https://mc-stan.org.

Cerquides, Jesus, Oguz Mulayim, Jerónimo Hernández-González, Amudha Ravi Shankar, and Jose Luis Fernandez-Marquez (2021). "A Conceptual Probabilistic Framework for Annotation Aggregation of Citizen Science Data". In: *Mathematics* 9.8. ISSN: 2227-7390. DOI: 10.3390/math9080875. URL: https://www.mdpi.com/2227-7390/9/8/875.

Dawid, Alexander and A. Skene (Jan. 1979). "Maximum Likelihood Estimation of Observer Error-Rates Using the EM Algorithm". In: *Applied Statistics* 28. DOI: 10.2307/2346806.

Doan, A., R. Ramakrishnan, and A. Halevy (2011). "Crowdsourcing systems on the World-Wide Web". In: *Communications of the ACM* 54, pp. 86 –96.

Downey, Allen (2015). *Bayesian Billiards*. URL: http://allendowney.blogspot.com/2015/06/bayesian-billiards.html.

Eddy, Sean R (2004). "What is Bayesian statistics?" In: *Nature Biotechnology* 22. DOI: 10.1038/nbt0904-1177. URL: https://www.nature.com/articles/nbt0904-1177.

Estellés-Arolas, Enrique and Fernando González L. Guevara (Apr. 2012). "Towards an Integrated Crowdsourcing Definition". In: *Journal of Information Science* 38. DOI: 10.1177/0165551512437638.

Frühwirth-Schnatter, Sylvia (2001). "Markov chain Monte Carlo Estimation of Classical and Dynamic Switching and Mixture Models". In: *Journal of the American Statistical Association* 96.453, pp. 194–209. DOI: 10.1198/016214501750333063. eprint: https://doi.org/10.1198/016214501750333063.

Heer, Jeffrey and Michael Bostock (2010). "Crowdsourcing Graphical Perception: Using Mechanical Turk to Assess Visualization Design". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '10. New York, NY, USA: Association for Computing Machinery, 203–212. ISBN: 9781605589299. DOI: 10.1145/1753326.1753357. URL: https://doi.org/10.1145/1753326.1753357.

Hernández-González, Jerónimo (2021). *Lecture notes in Probabilistic Graphical Models*.

Hooper, Martyn (2013). "Richard Price, Bayes' theorem, and God". In: *Significance* 10.1, pp. 36–39.

Howe, Jeff (2006). "The Rise of Crowdsourcing". In: URL: https://www.wired.com/2006/06/crowds/.

Koller, Daphne and Nir Friedman (Jan. 2009). *Probabilistic Graphical Models: Principles and Techniques*. ISBN: 978-0-262-01319-2.

Kullback, S. and R. A. Leibler (1951). "On Information and Sufficiency". In: *The Annals of Mathematical Statistics* 22.1, pp. 79 –86. DOI: 10.1214/aoms/1177729694. URL: https://doi.org/10.1214/aoms/1177729694.

La Vecchia, Gioacchino and Antonio Cisternino (2010). "Collaborative Workforce, Business Process Crowdsourcing as an Alternative of BPO". In: *Current Trends in Web Engineering*. Ed. by Florian Daniel and Federico Michele Facca. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 425–430. ISBN: 978-3-642-16985-4.

Lakhani, Karim, Lars Jeppesen, Peter Lohse, and Jill Panetta (Jan. 2007). "The Value of Openness in Scientific Problem Solving". In: *Harvard Business School WP, No. 07-050, 2007*.

Li, Hongwei (2015). "Theoretical Analysis and Efficient Algorithms for Crowdsourcing". PhD thesis. University of California, Berkeley, USA. URL: http://www.escholarship.org/uc/item/9rd5255z.

Lin, Jiayu (2016). "On The Dirichlet Distribution by Jiayu Lin". In: Queen's University.

Marin, Jean-Michel, Kerrie Mengersen, and Christian P. Robert (2005). "Bayesian Modelling and Inference on Mixtures of Distributions". In: *Bayesian Thinking*. Ed. by D.K. Dey and C.R. Rao. Vol. 25. Handbook of Statistics. Elsevier, pp. 459–507. DOI: https://doi.org/10.1016/S0169-7161(05)25016-2. URL: https://www.sciencedirect.com/science/article/pii/S0169716105250162.

McElreath, Richard (2018). *Algebra and the Missing Oxen*. Accessed: 2021-05-26. URL: https://elevanth.org/blog/2018/01/29/algebra-and-missingness/.

Nocedal, Jorge and Stephen J. Wright (2006). *Numerical Optimization*. second. New York, NY, USA: Springer.

Papastamoulis, Panagiotis (Feb. 2016). "label.switching: An R Package for Dealing with the Label Switching Problem in MCMC Outputs". In: *Journal of statistical software* 69, pp. 1–24. DOI: 10.18637/jss.v069.c01.

Passonneau, Rebecca J. and Bob Carpenter (2014). "The Benefits of a Model of Annotation". In: *Transactions of the Association for Computational Linguistics* 2, pp. 311–326. DOI: 10.1162/tacl_a_00185. URL: https://www.aclweb.org/anthology/Q14-1025.

Paun, Silviu, Bob Carpenter, Jon Chamberlain, Dirk Hovy, Udo Kruschwitz, and Massimo Poesio (Dec. 2018). "Comparing Bayesian Models of Annotation". In: *Transactions of the Association for Computational Linguistics* 6, pp. 571–585. ISSN: 2307-387X. DOI: 10.1162/tacl_a_00040. eprint: https://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00040/1567662/tacl_a_00040.pdf. URL: https://doi.org/10.1162/tacl_a_00040.

Rodríguez, Carlos E. and Stephen G. Walker (2014). "Label Switching in Bayesian Mixture Models: Deterministic Relabeling Strategies". In: *Journal of Computational and Graphical Statistics* 23.1, pp. 25–45. DOI: 10.1080/10618600.2012.735624. eprint: https://doi.org/10.1080/10618600.2012.735624.

Solomon Kullback Lindley, DV (1959). "Information Theory and Statistics". In: pp. 40–41. URL: http://index-of.co.uk/Information-Theory/Information\%20theory\%20and\%20statistics\%20-\%20Solomon\%20Kullback.pdf.

Stan Reference Manual. English. Version 2.27. Stan Development Team. 189 pp.

Stewart, Osamuyimen, Juan Huerta, and Melissa Sader (Jan. 2009). "Designing crowdsourcing community for the enterprise". In: pp. 50–53. DOI: 10.1145/1600150.1600168.

Vukovic, Maja, Mariana Lopez, and Jim Laredo (2010). "PeopleCloud for the Globally Integrated Enterprise". In: *Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops*. Ed. by Asit Dan, Frédéric Gittler, and Farouk Toumani. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 109–114. ISBN: 978-3-642-16132-2.

Walsh, Bruce (2004). "Markov chain monte carlo and gibbs sampling". In: *Lecture notes for EEB* 581.